SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2691

WEB I MOBILNO RJEŠENJE ZA RUKOVANJE TEMPERATURNIM LISTAMA ZASNOVANO NA TEHNOLOGIJI NFC

Mihovil Ćaćić

Zagreb, veljača 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2691

WEB I MOBILNO RJEŠENJE ZA RUKOVANJE TEMPERATURNIM LISTAMA ZASNOVANO NA TEHNOLOGIJI NFC

Mihovil Ćaćić

Zagreb, veljača 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 15. listopada 2021.

DIPLOMSKI ZADATAK br. 2691

Pristupnik:	**Mihovil Ćaćić (0036495899)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Alan Jović

Zadatak: **Web i mobilno rješenje za rukovanje temperaturnim listama zasnovano na tehnologiji NFC**

Opis zadatka:

Tehnologija NFC (engl. Near-Field Communication) danas se nalazi u gotovo svakom mobilnom uređaju, a najpoznatija je u kontekstu beskontaktnog plaćanja. U ovome diplomskom radu potrebno je pokazati praktičnu uporabu tehnologije NFC za rješavanje problema temperaturnih lista u bolničkom liječenju. Ideja je uvesti temperaturne liste u elektroničkom obliku na način gdje bi svaki bolnički krevet imao svoju NFC naljepnicu koju bi medicinsko osoblje skeniralo prilikom posjete te dobilo uvid u temperaturnu listu pacijenta. Po potrebi, podaci bi se mijenjali preko mobilne aplikacije. U diplomskom radu potrebno je razviti web i mobilno rješenje za upravljanje temperaturnim listama uz uporabu tehnologije NFC i simulaciju stvarnih bolničkih podataka. Web aplikacija koristila bi se za unos novih pacijenata, pregled temperaturnih lista i njihovo filtriranje po različitim parametrima. Mobilnu aplikaciju potrebno je ostvariti pomoću radnog okvira Flutter. I web i mobilna aplikacija trebaju podržati izbor predloška za temperaturne liste te njihov ispis u PDF format. U radu je potrebno istražiti i način povezivanja ovog rješenja s ostalim bolničkim uslugama i bazama podataka.

Rok za predaju rada: 4. veljače 2022.

Sadržaj

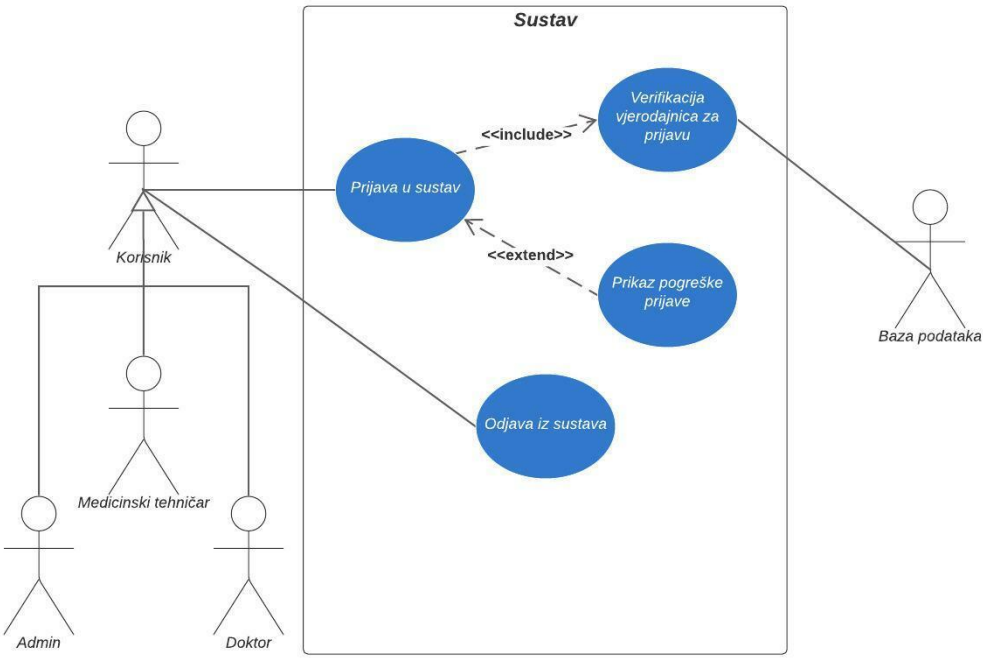
1. Uvod

Parametri pacijenta, poput temperature, krvnog tlaka, pulsa i glukoze u krvi, koji se svakodnevno prikupljaju tijekom njegovog boravka u bolnici, su važni pokazatelji pacijentovog zdravstvenog stanja, i dobri pokazatelji uspješnosti oporavka odnosno pogoršanja tog istog stanja. U većini zdrastvenih ustanova u Hrvatskoj, navedeni parametri pacijenta se zapisuju na temperaturne liste u papirnatom obliku koje se nalaze na bolničkim krevetima. Takav oblik temperaturnih lista nerijetko je slabo čitljiv, a nakon završetka hospitalizacije njihov sadržaj sa prikupljenim podacima se najčešće odlaže u bolničkim arhivama te ostaje zaboravljen. Pomno odabran dizajn temperaturne liste, tj. način prikaza prikupljenih podataka pacijenta, značajno pridonosi ranome otkrivanju pogoršanja njegovog stanja, što pak rezultira pravovremenom reakcijom medicinskih djelatnika [1]. Rješenje navedenih problema je elektronički zapis temperaturne liste, čime bi se podaci prikupljeni tijekom opservacije pacijenta spremali u bazu podataka. Prikaz tih podataka u digitalnom obliku omogućio bi medicinskom osoblju lakše očitavanje parametara a dobivene bi informacije bile lakše dostupne tijekom nastavka liječenja. Pristup temperaturnoj listi, prilikom vizite ili mjerenja parametara pacijenta, bio bi moguć korištenjem tehnologije NFC (engl. *Near-Field Communication*). Svaki bolnički krevet bi posjedovao oznaku zasnovanu na tehnologiji NFC (dalje: NFC-oznaka) čijim bi skeniranjem medicinski djelatnik dobio uvid u pacijentove podatke i po potrebi bi ih mijenjao unutar mobilne aplikacije.

2. Arhitektura i oblikovanje sustava

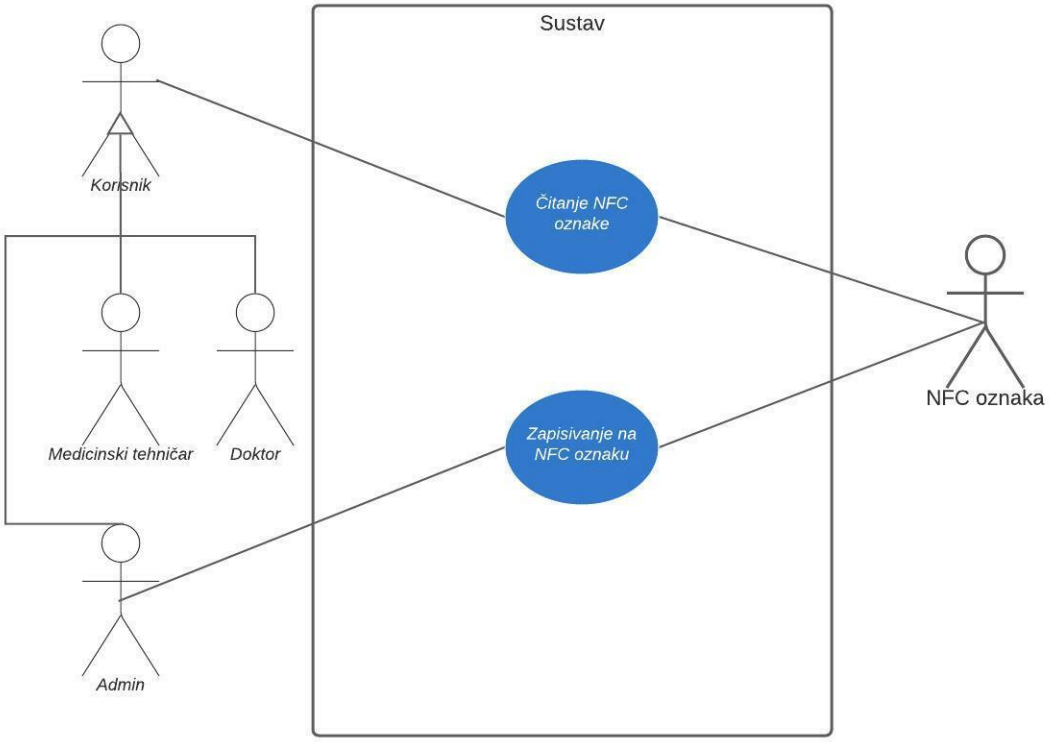
2.1. Funkcionalni zahtjevi sustava

U sustavu postoje tri vrste korisnika s različitim mogućnostima i dopuštenjima prilikom korištenja implementiranog rješenja. Svaki od njih prije pristupa funkcionalnostima sustava mora obaviti prijavu unosom korisničkog imena i lozinke, vidljivo na slici 1, nakon čega se obavljaju procesi autentifikacije i autorizacije. Identitet potvrđen autentifikacijom koristit će se prilikom svakog dodavanja i uređivanja podataka pacijenta, na način da će se uz svaki uneseni podatak bilježiti i identifikator korisnika koji je obavio unos. Time se jednoznačno određuje odgovornost korisnika prlikom akcija unutar sustava. Autorizacijom se određuje uloga korisnika, odnosno prava na pristup funkcionalnostima sustava koja su definirana dodijeljenom ulogom.



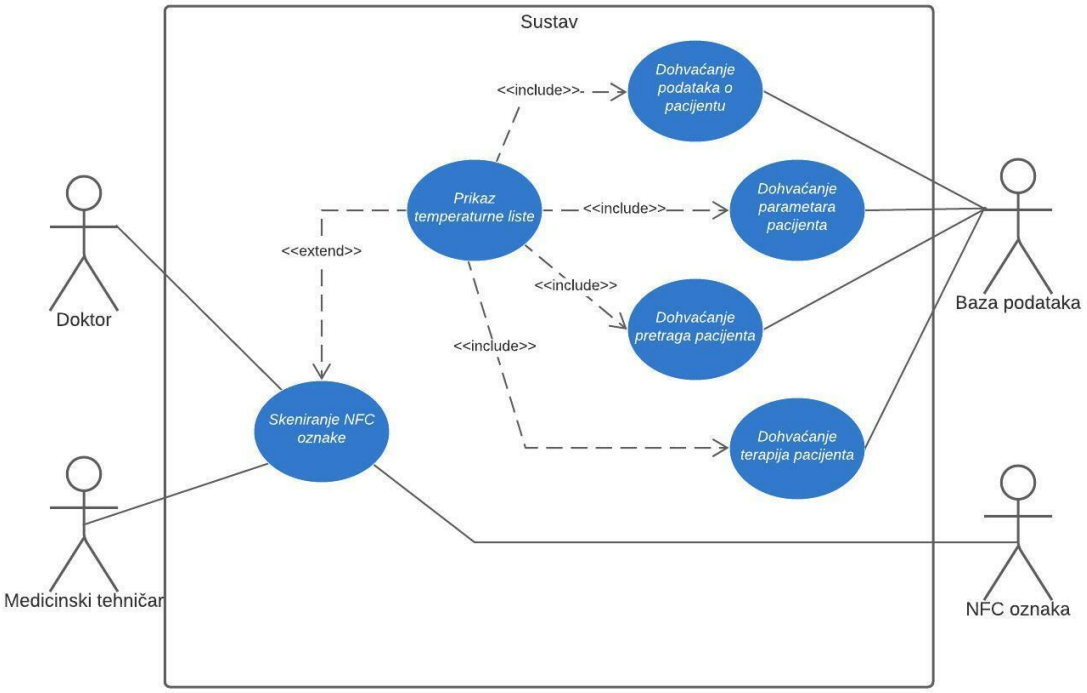
Slika 1. Obrazac uporabe – prijava u sustav

Na slici 2 vidimo grafički prikaz komunikacije mobilnog uređaja s NFC-oznakom koja se nalazi na bolničkom krevetu. Svi korisnici imaju mogućnost skeniranja, odnosno čitanja NFC-oznake, dok samo korisnik s ulogom administratora (admin) ima mogućnost zapisivanja podataka na NFC oznaku. Akcije prilikom čitanja podataka s NFC-oznake su opisane u nastavku.



Slika 2. Obrazac uporabe – komunikacija s NFC-oznakom

Korisnici s ulogama admin ili doktor, korištenjem mobilnog uređaja koji podržava rad s tehnologijom NFC, imaju mogućnost pristupa temperaturnoj listi pacijenta skeniranjem NFC-oznake na bolničkom krevtu. Takva oznaka sadrži identifikator bolničkog kreveta koji je u sustavu povezan s trenutnom hospitalizacijom pacijenta. Njezinim skeniranjem mobilni uređaj dohvaća informaciju o navedenom identifikatoru te na temelju njega iz baze podataka dohvaća osobne podatke o pacijentu, opservacijske podatke prikupljene tijekom njegovog boravka u bolnici (temperatura, krvni tlak, puls itd.), ali i ostale medicinske informacije bitne za liječenje (pretrage i terapije).



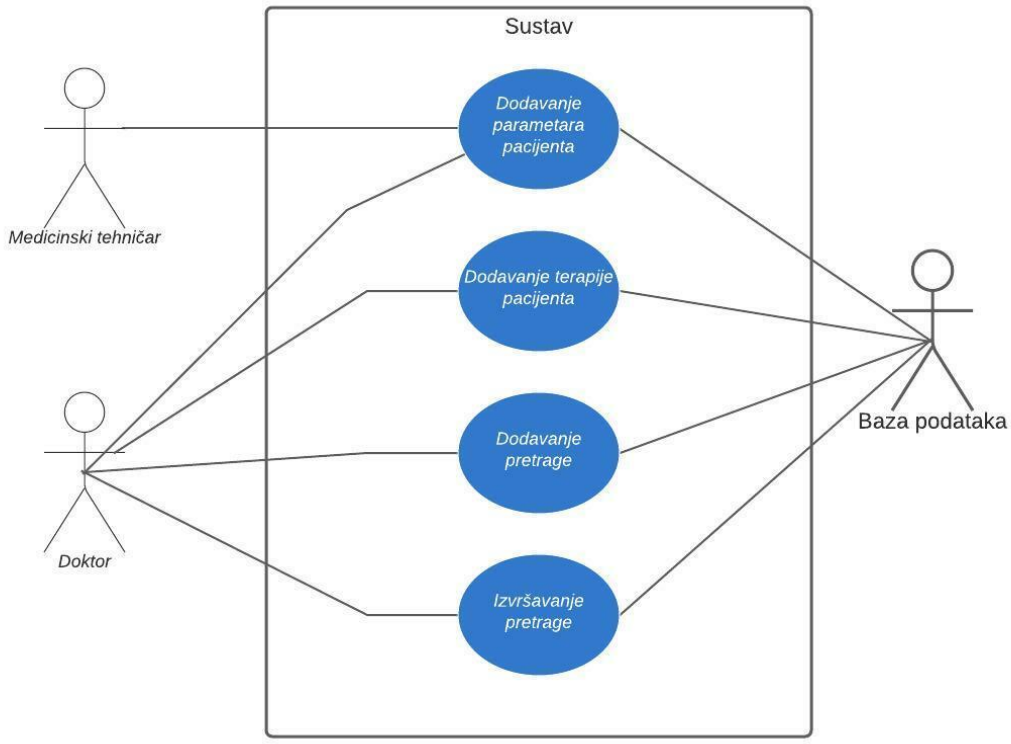
Slika 3 Obrazac uporabe – prikaz temperaturne liste pacijenta

Informacije prikazane na temperaturnoj listi pacijenta možemo podijeliti u nekoliko kategorija. Prva kategorija su osobni podaci o pacijentu, kao što su ime i prezime, datum rođenja, matični broj osiguranika, itd. Navedeni podaci se ne mogu unositi i izmjenjivati, već se oni dohvaćaju prilikom hospitalizacije iz baze podataka svih pacijenata u zdravstvenom sustavu. Za potrebe izrade diplomskog rada, baza podataka svih pacijenata je implementirana kao relacija unutar iste baze podataka u kojoj se nalaze ostali podaci o temperaturnoj listi. U praksi bi se dohvat podataka o pacijentu obavio direktnim spajanjem na bazu postojećeg zdravstvenog sustava ili korištenjem web usluge (eng. REST *web service*) zdravstvenog sustava.

Uz osnovne informacije još su nam prikazani parametri pacijenta, terapije i pretrage. Na slici 4, vidimo da korisnici s ulogama doktor i medicinski tehničar imaju mogućnost unosa parametara pacijenta. Parametri su opservacijski podaci (temperatura, krvni tlak itd.) koji se svakodnevno unose prilikom mjerenja. Njih najćešće unosi medicinski tehničar, a vrlo je bitna njihova vremenska komponenta koja nam pokazuje stanje pacijenta tijekom proteklog vremena. Prilikom dodavanja parametara potrebno je unijeti podatke o temperaturi pacijenta, krvnom tlaku, diurezi, stolici, glukozi u krvi i datumu prilikom kojeg je odrađeno mjerenje.

Mogućnost dodavanja terapija ima samo korisnik s ulogom doktor. Prilikom unosa, potrebno je odabrati raspon datuma pri kojem će se terapija pružati, naziv terapije, i količinu za svaki dio dana (jutro, podne, večer).

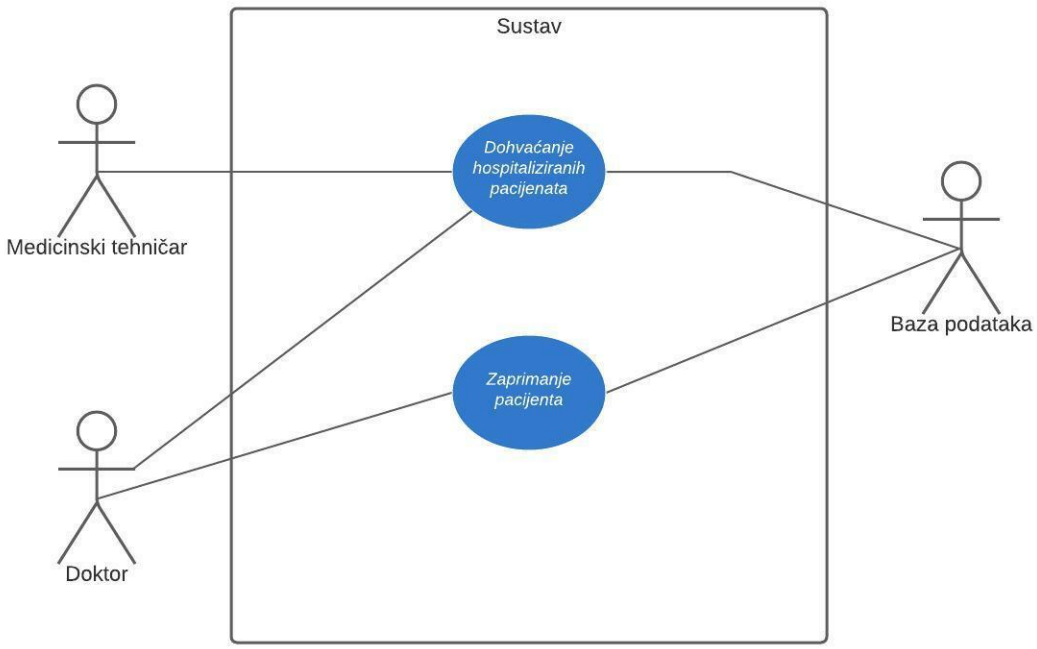
Pretrage pacijenta također su dio temperaturne liste. Na akciju dodavanja pretrage korisnik unosi njezin naziv i datum izvršavanja. Neposredno nakon dodavanja, status pretrage je „ne izvršen“, te korisnik s ulogom doktor ima mogućnost odabira pojedine pretrage i promjene njezinog statusa.



Slika 4 Obrazac uporabe – unos podataka u temperaturnu listu

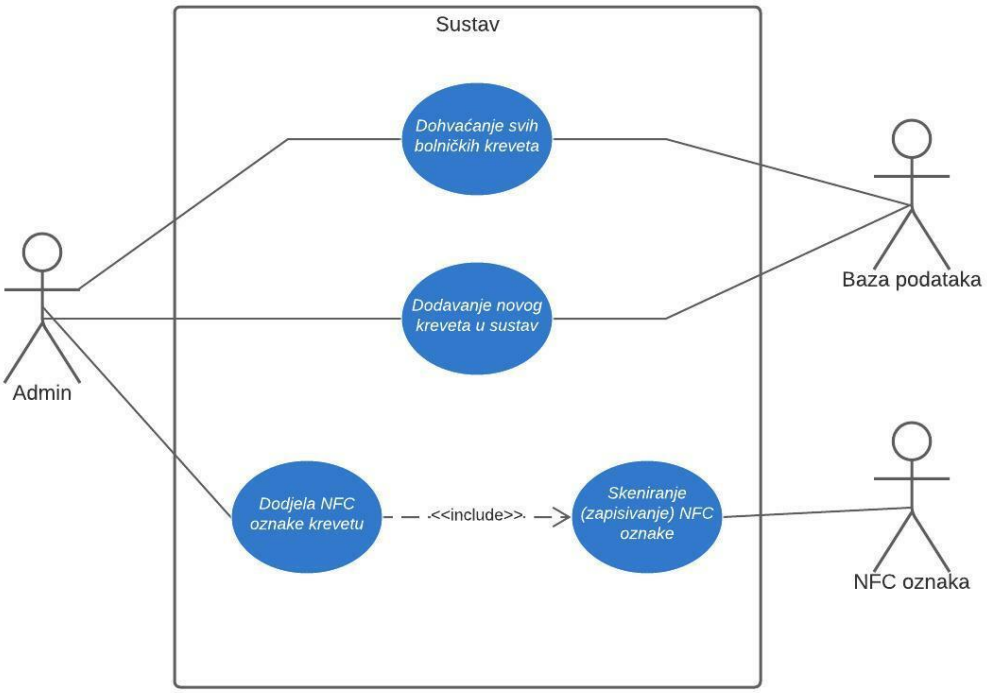
Unutar web aplikacije, na početnom zaslonu, korisnici s ulogama medicinski tehničar i doktor imaju vidljiv tablični prikaz svih trenutno hospitaliziranih pacijenata. Odabirom pojedinog zapisa iz tablice otvara se temperaturna lista pacijenta sa svim podacima kao u prethodno opisanoj funkcionalnosti dohvaćanja temperaturne liste nakon skeniranja NFC-oznake.

Uz prikaz svih pacijenata, korisnik s ulogom doktor ima i mogućnost zaprimanja pacijenta na odjel. Prilikom zaprimanja, korisnik mora odabrati pacijenta odabirom vrijednosti iz liste svih pacijenata u zdravstvenom sustavu i jedan od slobodnih kreveta na kojeg želi smjestiti pacijenta. Nakon spremanja unesenih podataka, odabrani krevet je povezan s pacijentom te će skeniranje NFC oznake tog kreveta rezultirati prikazom temperaturne liste zaprimljenog pacijenta. Obrazac uporabe opisanih funkcionalnosti vidljiv je na slici 5.



Slika 5 Obrazac uporabe – hospitalizacija pacijenta

Na slici 6 vidljive su mogućnosti korisnika s ulogom admin vezane uz rukovanje bolničkim krevetima. Nakon prijave u sustav, korisnik ima tablični prikaz svih kreveta koji se trenutno nalaze na odjelu. Prilikom dodavanja novog kreveta u sustav, korisnik mora unijeti podatke o serijskom broju kreveta te sobi unutar odjela u kojoj će se taj krevet nalaziti. Nakon dodavanja, krevetu je dodijeljena jedinstvena oznaka unutar sustava te korisnik unutar mobilne aplikacije ima mogućnost odabira novododanog kreveta i skeniranjem NFC-oznake zapisuje jedinstveni identifikator na samu oznaku. Time je novi krevet spreman za korištenje i smještanje pacijenta.

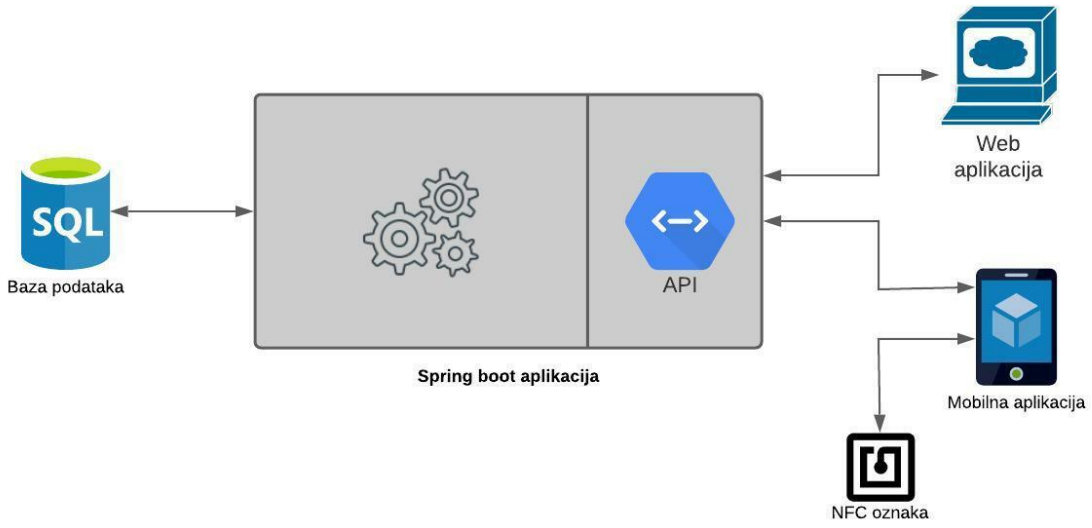


Slika 6 Obrazac uporabe – rukovanje bolničkim krevetima

2.2. Opis sustava

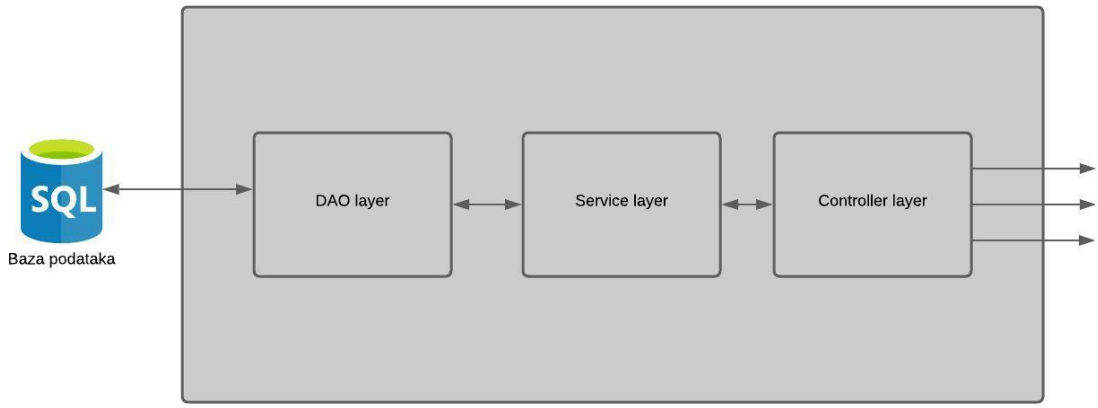
Glavni dio sustava implementiran je kao web usluga koja prati arhitekturni stil REST (engl. *REpresentational State Transfer*), vidljivo na slici 7. Takav stil arhitekture zahtijeva komunikaciju između klijenta i poslužitelja pomoću HTTP protokola s jasno propisanim sučeljem u kojem su definirani ulazni resursi (engl. *request resources*) poslani od strane klijenta prema poslužitelju, izlazni resursi (engl. *response resources*) kao poslužiteljev odgovor klijentu i zaglavlja HTTP-zahtjeva s odgovarajućom HTTP-metodom (GET, POST, PUT, DELETE). Prateći jasno definirano sučelje, postiže se neovisnost između klijentske i poslužiteljske strane. Svaki poslani HTTP-zahtjev je neovisan o ostalim zahtijevima poslanima prema sustavu, što je također jedno od pravila REST api stila arhitekture (eng. stateless) [2]. Svi HTTP odgovori unutar aplikacije su označeni kao odgovori koji nisu pogodni za spremanje u priručnu memoriju. Dio sustava koji implementira web usluge je podijeljen po slojevima, a njihova međusobna komunikacije je opisana u nastavku.

Dio sustava na poslužiteljskoj strani, nakon primljenog HTTP-zahtijeva, obavlja niz akcija te po potrebi dohvaća i sprema podatke u bazu podataka. Mobilna aplikacija, uz mogućost pristupa web uslugama na poslužiteljskom dijelu ima mogućnost i komunikacije s NFC-oznakama ukoliko mobilni uređaj podržava komunikaciju putem NFC protokola.



Slika 7 Pregled ostvarenog programskog sustava

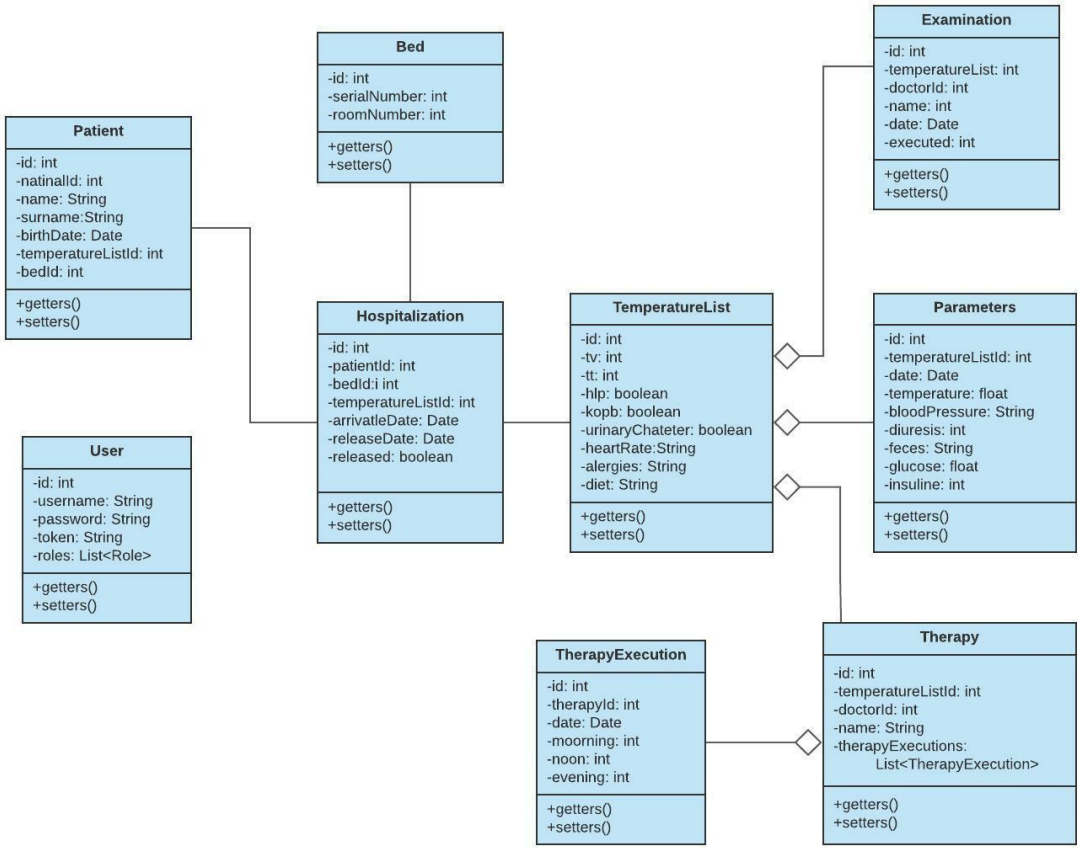
Na slici 8 vidimo da je poslužiteljski dio sustava podjeljen u nekoliko slojeva koji su međusbno neovisni, a komuniciraju putem jasno definiranih metoda. Tako imamo sloj nadglednika (engl. *controller layer*) koji je zadužen za primanje zahtjeva i slanje odgovora prema klijentskoj strani. Njegovim metodama i argumentima metoda su definirani ulazni i izlazni parametri prilikom HTTP zahtjeva zaprimljenih s klijentske strane. Sloj nadglednika provjerava ispravnost primljenog zahtjeva, što podrazumijeva ispravnu HTTP-metodu, ulazne parametre i vrijednosti zaglavlja HTTP-zahtjeva, nakon čega poziva odgovarajuću metodu sloja usluge (engl. *service layer*) te mu prosljeđuje dobivene parametre. Unutar sloja usluge obavlja se poslovna logika sustava, a on vidi i radi s podacima iz baze koje je dobio od podatkovnog sloja kao i s ulaznim resursima primljenim od klijenta. Podatkovni sloj je zadužen za unos i dohvaćanje podataka iz baze. Njegova implementacija je neovisna o sloju usluge, a komuniciraju putem sučelja koji podatkovni sloj implementira [3]. Komunikacija između baze podataka i podatkovnog sloja je definirana JDBC-sučeljem (engl. Java Database Connectivity API), a pružatelji usluga baza podataka su dužni pružiti vlastitu implementaciju programa koji implementira JDBC API. Takav program se naziva upravljački program JDBC-a (engl. JDBC *driver*), a omogućuje izvršavanje upita nad bazom podataka iz podatkovnog sloja [4].



Slika 8 Arhitektura poslužiteljskog dijela ostvarenog sustava

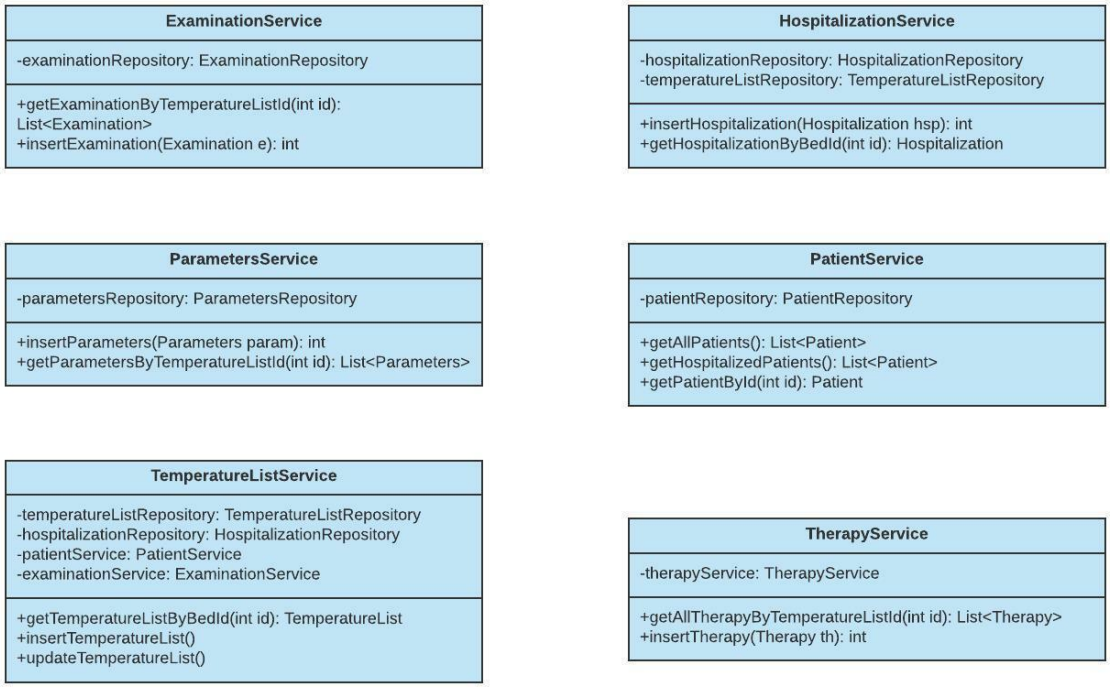
2.3. Dijagram razreda

Slika 9 prikazuje dijagram razreda, koji čine skup modela unutar našeg sustava koji su međusobno povezani. Razred **Hospitalization** nam modelira boravak pacijenta u bolnici. On je povezan s razredom **Patient**, koji nam pruža informaciju o kojem se pacijentu radi, razredu **Bed**, koji nam govori na kojem krevetu je smješten pacijent, a čije će nam informacije biti bitne za rukovanje s NFC-oznakama, i razredu **TemperatureList**. Razred **TemperatureList**, s članskim varijablama **tt, tv, kopb, heartRate, alergies, diet** i dr. čije se vrijednosti najčešće definiraju na početku hospitalizacije i nemaju vlastitu vremensku komponentu nam predstavlja glavni entitet u sustavu na koji se veže nekoliko lakših entiteta, **Examination, Parameters, Therapy**. Navedena tri entitea imaju bitnu vremensku komponentu koja nam, kod entiteta **Parameters** pruža informaciju o promjeni vrijednosti parametara pacijenta (temperatura, krvni talk itd.) tijekom vremena, a kod entiteta **Examination** i **Therapy** informaciju o vremenu izvršavanja pojedine pretrage, odnosno vremenu davanja određene terapije.



Slika 9 Dijagram razreda – modeli

Na slici 10 su prikazani razredi sloja usluge koji se jednom riječju nazivaju usluge (engl. *service*). Sadrže članske varijable koje koje su označene kao repozitoriji i koje su dio podatkovnog sloja, kao i instance drugih razreda označenih kao usluge. Unutar njihovih metoda su implementirane funkcionalnosti koje nam sustav pruža, ali neke sigurnosne akcije, kao naprimjer provjera ispravnosti ulaznih podataka i provjera prava pristupa korisnika na pojedinu funkcionalnost.



Slika 10 Dijagram razreda – usluge

3. Korištene tehnologije i alati

3.1. Spring

Za izradu poslužiteljskog dijela sustava korišten je radni okvir Spring i alat Spring Boot koji nam omogućava pokretanje aplikacije uz jednostavno postavljanje konfiguracijskih postavki. Upravo to ga čini popularnim i često korištenim radnim okvirom za izradu web aplikacija, ali i drugih oblika programskih rješenja kao što su aplikacije u obaku, mikro servisa i dr. Spring Boot podržava korištenje alata Maven ili Gradle za izradu programa (engl. *build tool*), a u implementaciji rješenja korišten je alat Gradle sa verzijom 7.3. Prema zadanim postavkama Spring Boota, korišten je web kontejner Tomcat (engl. Tomcat *web container*) unutar kojeg se izvodi web aplikacija, a zadužen je za rukovanje HTTP-zahtijevima pristiglima na poslužitelj. Uz njega, Spring Boot podržava i korištenje web okvira Jetty i Undertow [5].

Spring koristi tehniku ubacivanja ovisnosti (engl. *dependency injection*) koja je jedan od oblika tehnike inverzije upravljanja (engl. *inversion of control*). Korištenjem takve tehnike naša aplikacije nema potrebu stvarati vlastite instance određenih razreda, već su one dohvaćene od strane radnog okvira. Za kreiranje instanci takvih razreda i njihov životni ciklus zadužen je Spring IoC Container. Spring nam omogućuje da kreiramo vlastite razrede kojima na jednom mjestu definiramo način na koji će se kreirati njegova instanca, takve razrede je potrebno označiti anotacijom @Bean. Spring IoC Container će stvoriti instancu takvog razreda prilikom pokretanja aplikacije ili prilikom potrebe korištenja razreda, a dalje u programu prilikom svake uporabe definiranog razreda koristit će se jedna te ista njegova instanca [6].

3.2. Flutter

Za izradu mobilne aplikacije korišten je alat Flutter koji je razvio Google. Naziv Flutter je zajednički naziv za Flutterov radni okvir i Flutterov SDK (engl. *Software Development Kit*). On nam omogućuje izradu višeplatformskih aplikacija (Android, iOS, Windows itd.) iz istog programskog koda. Pomoću Flutterovog SDK-a napisani programski kod se prevodi u nativni izvršni kod. Pri tome valja napomenuti da Flutter ne prevodi kod u međukod koji se zatim pomoću klasičnih alata, naprimjer Androidovog SDK-a, prevodi u izvršni kod, već ga Flutter u potpunosti prevodi u strojni kod željene platforme. Flutterov radni okvir nam omogućuje korištenje gotovih komponenti korisničkog sučelja, ali i razrede za rad s modulima mobilnih uređaja kao što su: kamera, čitač otiska prsta, modul NFC i sl. Prilikom pokretanja Flutterove aplikacije, operacijski sustav platforme aplikaciji omogućuje upravljanje platnom (engl. *canvas*) odnosno daje joj mogućnost prikaza svojih komponenti na zaslonu i pristup događajima (detekcija dodira zaslona) i uslugama, kao što su kamera ili lokacijske usluge [7].

Za razvoj aplikacija Flutter koristi se programski jezik Dart, također razvijen od Googlea. Dart je objektno orijentirani programski jezik koji ima mogućnost prevođenja u nativni strojni kod platforme i u JavaScript kod za izvršavanje unutar web preglednika. Programski kod napisan u Flutteru se može prevoditi na dva načina. Prevođenjem prije izvršavanja (engl. *ahead of time compile*) način je koji se koristi prilikom puštanja u rad gotove aplikacije na pojedinu platformu, i prevođenje tijekom izvršavanja (engl. *just in time compile*) koji nam olakšava razvoj aplikacija tako da prilikom promjena dijela koda nije potrebno ponovno prevođenje cijelog programskog koda i njegovo pokretanje na platformi, već samo prevođenje promijenjenog dijela koda što nam uvelike ubrzava proces razvoja

4. Implementacija sustava

4.1. Implementacija sustava na poslužiteljskoj strani

Za izgradnju dijela sustava na poslužiteljskoj strani korišteni su određeni moduli radnog okvira Spring. Modul ***Spring Boot Starter Web*** je zadužen za pokretanje aplikacije unutar web kontejnera Tomcat i način definiranja krajnjih točaka web aplikacije pomoću kontrolera. Za mogućnost komunikacije s bazom podatka i izvršavanjem upita iz podatkovnog sloja zadužen je modul ***Spring boot jdbc***. Modul ***Spring security*** nam nudi razrede koje nam omogućuju implementaciju sigurnosnih zahtijeva, naprimjer definira sigurnosni kontekst aplikacije pomoću kojega obavljamo procese autorizacije. Za upravljanje, generiranje i provjeru ispravnosti korisničkog tokena korišten je modul ***Jjwt***.

Spring boot koristi metodu automatske konfiguracije, što znači da na temelju modula koje koristimo automatski postavlja određene zadane vrijednost. Naprimjer, ako smo definirali korištenje modula ***Hsqldb*** a nismo postavili vrijednosti za uspostavljanje veze na bazu podataka, Spring će sam konfigurirati i stvoriti instancu baze podataka u privremenoj memoriji [5]. Kako bi nadjačali navedenu konfiguraciju, dovoljno je samo postaviti vrijednosti potrebne za spajanje na bazu unutar datoteke ***Application.properties*** ili unutar konfiguracijskog razreda stvoriti metodu označenu anotacijom @Bean koja definira instancu traženog razreda, u ovom slučaju razreda `DataSource`.

Primjer nadjačavanja konfiguracijskih postavki vidljiv je na slici 11. Razred `DaoAuthenticationProvider` koristi se prilikom izvođenja funkcionalnosti vezanih uz prijavu korisnika. Definiranjem ovakve metode uz anotaciju @Bean, navodimo Spring da prilikom svakog korištenja spomenutog razreda koristi instancu čije smo članske varijable `userDetailService` i `passwordEncoder` postavili na željene vrijednosti.

```java
@Bean
public DaoAuthenticationProvider authenticationProvider() {
    final DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userService);
    authProvider.setPasswordEncoder(passwordEncoder());
    return authProvider;
}
```

Slika 11. Primjer definiranja razreda @Bean

Na slici 12 vidimo metodu koja se izvršava prilikom prijave korisnika u sustav. Argument metode `request` sadrži podatke koje je korisnik unio prilikom prijave (korisničko ime i loznika). Pomoću objekta `authenticationManager`, koji je dio modula ***Spring security***, provjeravamo ispravnost unesenih podataka i identificiramo korisnika. Ako se autentifikacija uspješno izvršila, na temelju korisničkih podataka generiramo token koji vraćamo kao vrijednost unutar HTTP-zaglavlja, a koji će klijentska strana dalje koristiti prilikom slanja HTTP-zahtjeva za pristup ostalim funkcionalnostima sustava.

```java
@PostMapping("login")
public ResponseEntity<User> login(@RequestBody AuthRequest request){
    Authentication authenticate = authenticationManager
            .authenticate(
                new UsernamePasswordAuthenticationToken(
                    request.getUsername(), request.getPassword()
                )
            );
    User user = (User) authenticate.getPrincipal();
    String token=jwtTokenUtil.generateAccessToken(user);
    user.setToken(token);
    return ResponseEntity.ok()
            .header(
                HttpHeaders.AUTHORIZATION,
                token
            )
            .body(user);
}
```

Slika 12 Metoda za prijavu u sustav

Prilikom svakog HTTP-zahtjeva, prije nego što se pozove određena funkcija odgovarajućeg nadglednika, potrebno je provjeriti ispravnost korisničkog tokena. Taj postupak je vidljiv na slici 13. Iz zaglavlja HTTP-zahtjeva dohvaćamo vrijednost poslanog tokena, a njegovu ispravnost utvrđujemo metodama koje nam pruža modul ***Jjwt***. S obzirom na to da smo prilikom prijave generirali token na temelju korisničkih podataka, sada možemo na temelju primljenog tokena identificirati korisnika. Korisničke vrijednosti spremamo u sigurnosni kontekst (engl. *security context*) koji ćemo kasnije u sloju usluga koristiti prilikom provjere prava pristupa na određenu funkcionalnost.

```java
@Component
@RequiredArgsConstructor
public class JwtTokenFilter extends OncePerRequestFilter {

    private final JwtTokenUtil jwtTokenUtil;
    private final UserService userService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {

        final String header = request.getHeader(HttpHeaders.AUTHORIZATION);
        if (isEmpty(header)|| !header.startsWith("Bearer ")) {
            chain.doFilter(request, response);
            return;
        }

        final String token = header.split(" ")[1].trim();
        if (!jwtTokenUtil.validate(token)) {
            chain.doFilter(request, response);
            return;
        }

        UserDetails userDetails = userService
                .loadUserByUsername(jwtTokenUtil.getUsername(token));

        UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(
                userDetails, null,
                ofNullable(userDetails).map(UserDetails::getAuthorities).orElse(of())
        );

        authentication
                .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        chain.doFilter(request, response);

    }
```

Slika 13 Razred JwtTokenFilter

Na slici 14 vidimo primjer razreda koji je označen kao kontroler (@RestController). Primanjem određenog HTTP-zahtjeva poziva se metoda odgovarajućeg kontrolera. Za to se brine komponenta ***Spring Dispacher Servlet*** koja preslikava primljeni HTTP zahtjev na odgovarajuću metodu nekog razreda. Valja napomenuti da se Spring također brine da je poslana ispravna vrijednost HTTP-metode i da se primljeni resurs unutar tijela HTTP-zahtjeva može mapirati u objekt odgovarajuće klase koja odgovara argumentu metode kontrolera. Nakon uspješno primljenog HTTP-zahtjeva kontroler poziva odgovarajuće metode sloja usluge.

```
@RestController
public class TemperatureListController {
    private final TemperatureListService temperatureListService;

    public TemperatureListController(TemperatureListService temperatureListService) {
        this.temperatureListService=temperatureListService;
    }

    @GetMapping("/getTemperatureListByBedId")
    public ResponseEntity<TemperatureListView> getTemperatureListByBedId(@RequestParam Integer bedId) throws Exception{
        return new ResponseEntity<>(temperatureListService.getTemperatureListViewByBedId(bedId),HttpStatus.OK);
    }

    @PostMapping("/updateTemperatureList")
        public Integer updateTemperatureList(@RequestBody TemperatureList temperatureList) {
            return temperatureListService.updateTemperatureList(temperatureList);
        }

}
```

Slika 14 Razred TemperatureListController

Na slici 15 vidimo implementaciju razreda `HospitalizationService` s metodom `insertHospitalization` koja je zadužena za obavljanje funkcionalnosti prilikom prijema novog pacijenta na odjel. Prvi korak unutar spomenute metode je provjera prava pristupa na traženu funkcionalnost. Taj postupak obavljamo pomoću razreda `SecurityContextHolder` koji nam omogućuje pristup sigurnosnom kontekstu u koji smo spremili podatke o korisniku dobivene prilikom provjere korisničkog tokena. U navedenom primjeru korisnik mora sadržavati ulogu doktor kako bi se obavila funkcionalnost prijema pacijenta na odjel. Nakon uspješne provjere prava na pristup funkcionalnosti, provjeravamo je li moguće izvršiti željenu hospitalizaciju, tj. je li krevet na koji se želi smjestiti novog pacijenta već zauzet. Ako je željena hospitalizacija moguća, postavljamo vrijednost članske varijable `ArrivalDate` na datum unosa nakon čega pozivamo metodu podatkovnog sloja.

```java
@Service
@RequiredArgsConstructor
public class HospitalizationService {
    private final HospitalizationRepository hospitalizationRepository;
    private final TemperatureListService temperatureListService;

    @Transactional
    public Integer insertHospitalization(Hospitalization hospitalization) throws Exception {
        if(SecurityContextHolder.getContext().getAuthentication().getAuthorities().contains(new Role("DOCTOR"))) {

            if(!hospitalizationRepository.isBedFree(hospitalization.getBedId())) {
                return null;
            }

            hospitalization.setArrivalDate(new Date());
            hospitalization.setReleased(false);
            hospitalization.setTemperatureListId(temperatureListService.insertTemperatureList());

            return hospitalizationRepository.insertHospitalization(hospitalization);
        }
        return null;
    }

    public Hospitalization getHospitalizatonByBedId(Integer bedId) {
        return hospitalizationRepository.getHospitalizationByBedId(bedId);
    }
}
```

Slika 15 Razred HospitalizationService

Unutar razreda `HospitalizationRepository` vidljivog na slici 16 obavlja se spremanje podataka o hospitalizaciji u odgovarajuću relaciju baze podataka. Razredi `NamedParameterJdbcTemplate`, `MapSqlParameterSource` i `KeyHolder` dijelovi su modula ***Spring Jdbc*** a služe nam za izvršavanje napisanog upita nad bazom podataka. Pomoću razreda `MapSqlParameterSource` definiramo vrijednosti parametara koje smo naveli unutar upita kojeg želimo izvršiti.

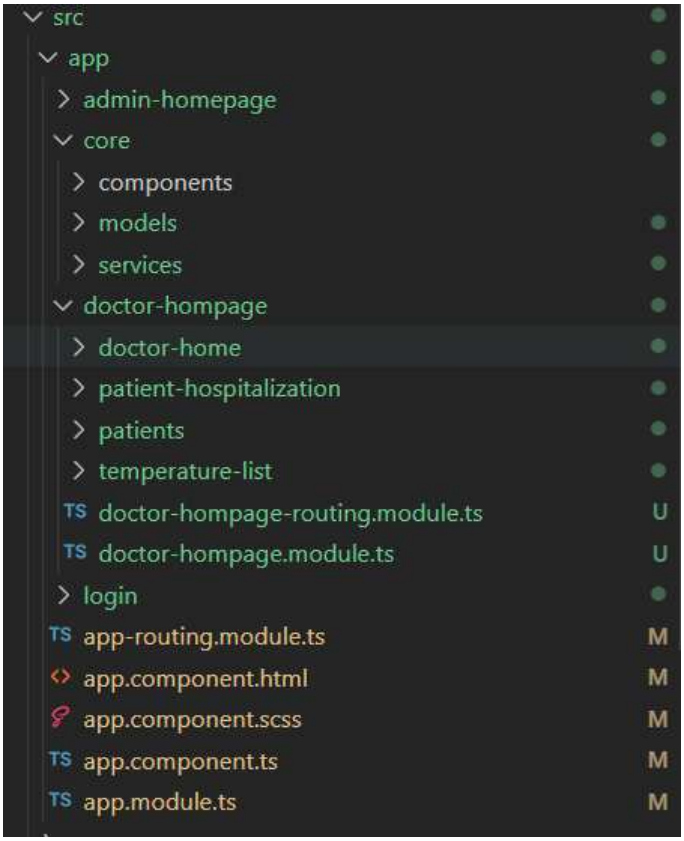
```java
@Repository
public class HospitalizationRepository {
    private final NamedParameterJdbcTemplate njdbc;

    public HospitalizationRepository(NamedParameterJdbcTemplate njdbc) {
        this.njdbc=njdbc;
    }

    public Integer insertHospitalization(Hospitalization hospitalization) {
        StringBuilder query = new StringBuilder();
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        KeyHolder keyHolder = new GeneratedKeyHolder();

        query.append("INSERT INTO hospitalization(patient_id, bed_id, temperature_list_id, arrival_date, released)" );
        query.append("VALUES (:patientId, :bedId, :temperatureListId, :arrivalDate, :released)");

        parameters.addValue("patientId", hospitalization.getPatientId());
        parameters.addValue("bedId", hospitalization.getBedId());
        parameters.addValue("temperatureListId", hospitalization.getTemperatureListId());
        parameters.addValue("arrivalDate", hospitalization.getArrivalDate());
        parameters.addValue("released", hospitalization.getReleased());

        njdbc.update(query.toString(),parameters, keyHolder,new String[] {"id"});

        return keyHolder.getKeyAs(Integer.class);
    }
```

Slika 16 Razred HospitalizationRepository

Prilikom dodavanja vrijednosti parametara sql upita Spring se brine o njihovoj ispravnosti. Tako smo sigurni od pokušaja napada sql ubrizgavanjem (eng. *sql injection*). Razred ***KeyHolder*** nam služi kako bi nakon izvršenog upita i uspješnog dodavanja zapisa u relaciju baze podataka imali vrijednost identifikatora unesenog retka. ***NamedParameterJdbcTemplate*** pomoću metode ***update*** izvršava željeni upit nad bazom podataka.

4.2. Implementacija web aplikacije

Prilikom izrade web aplikacije korišten je radni okvir Angular a njezinu implementaciju smo podijelili u nekoliko modula. Prvi je modul ***login*** koji je zadužen za prijavu korisnika i određivanja korisnikove uloge na temelju koje prikazujemo odgovarajući model i njegove komponente. Modul ***admin-hompage*** je zadužen za prikaz komponenti koje implementiraju rukovanje krevetima, dok je ***doctor-homepage*** modul zadužen za prikaz komponenti vezanih uz rukovanje temperaturnim listama pacijenata. Na slici 17 vidimo hijerarhijski prikaz modula i komponenti unutar Angularove aplikacije.



Slika 17 Hijerarhijski prikaz modula i komponenti

Unutar mape ***core*** definirali smo modele koji nam modeliraju podatke dobivene od strane poslužiteljskoj dijela sustava i razred `HttpService` u kojem smo implementirali HTTP pozive prema poslužitelju. Vidljivo na slici 18.

```
export class HttpService {

  constructor(private http: HttpClient) { }


  getHttp(apiUrl: string, data?: any, headers?: any) {
    return this.http.get<any>(`http://localhost:8080/${apiUrl}`, { headers: headers, params: data });
  }

  postHttp(apiUrl: string, data?: any, headers?: any) {
    return this.http.post<any>(`http://localhost:8080/${apiUrl}`, data, { headers: headers});
  }

  putHttp(apiUrl: string, data?: any, options?: any) {
    return this.http.put<any>(`http://localhost:8080/${apiUrl}`, data, options);
  }

  deleteHttp(apiUrl: string, data?: any) {
    return this.http.delete<any>(`http://localhost:8080/${apiUrl}`, { body: data });
  }

}
```

Slika 18 Razred HttpService

Modul ***login*** sadrži komponentu `LoginComponent` s metodom `login` koja šalje HTTP-zahtjev na poslužitelj zajedno s unesenim podacima korisnika (korisničko ime i lozinka). Nakon uspješne prijave u sustav, korisnički token se sprema u lokalnu memoriju web preglednika kako bi se mogli obavljati ostali HTTP pozivi unutar aplikacije. Komponenta na temelju dobivenog odgovora od poslužitelja uz korisnički token dobiva i informaciju u ulozi prijavljenog korisnika i na temelju nje usmjerava korisnika na odgovarajuću rutu. Prikaz implementacije komponente `LoginComponent` prikazan je na slici 19.

```
export class LoginComponent implements OnInit {

  public loggedIn : Boolean = false;

  constructor(
    private httpService : HttpService,
    private router: Router){}

  ngOnInit(): void {
  }

  login(){
    let user={
      "username": (document.getElementById("username") as any).value,
      "password": (document.getElementById("password") as any).value
    }
    this.httpService.postHttp('public/login',user).subscribe((resposne : any)=>{
      if(resposne.token){
        this.loggedIn=true;
        localStorage.setItem('token',resposne.token);
        localStorage.setItem('role',resposne.authorities[0].name)
        if(resposne.authorities[0].name ==='DOCTOR'){
          this.router.navigate(['/doctorHomepage']);
        }else if(resposne.authorities[0].name ==='ADMIN'){
          this.router.navigate(['/adminHomepage']);
        }

      }
    })

}
```

Slika 19 Komponenta LoginComponent

Usmjeravanjem na rutu „/doctorHompage“ prikazuje nam se komponenta `DoctorHomeComponent` koja u sebi sadrži komponentu `PatientComponent`. Unutar komponente `PatientComponent` se prilikom njezinog kreiranja, odnosno poziva metode `ngOnInit`, dohvaćaju svi hospitalizirani pacijenti. Na slici 20 vidimo da prije slanja HTTP-zahtjeva za dohvat svih hospitaliziranih pacijenata, moramo iz lokalne memorije web preglednika dohvatiti korisnički token koji smo pohranili prilikom korisnikove prijave u sustav. Valja napomenuti da ovakav način spremanja korisničkog tokena onemogućuje čitanje njegove vrijednosti od strane drugih aplikacija unutar istog web preglednika te na taj način spriječavamo zlonamjerne HTTP-pozive od drugih aplikacija.

```
getHospitalizedPatients(){
  let token = localStorage.getItem('token');
  let headers = new HttpHeaders({'Authorization': 'Bearer ' + token})
  this.httpService.getHttp('getHospitalizedPatients', null, headers).subscribe(response => {
    this.dataSource=response;
  });
}
```

Slika 20 Metoda za dohvat hospitaliziranih pacijenata

Usmjeravanjem na rutu „/temperatureList/bed?bedId“, koje se odvija prilikom odabira pojedinog pacijenta iz liste hospitaliziranih pacijenata, prikazuje nam se komponenta `TemperatureListComponent`. Ona je zadužena za dohvaćanje podataka temperaturne liste pacijenta, a dobivene podatke prosljeđuje komponentama `TherapiesComponent`, `ParametersComponent` i `ExaminationComponent`, vidljivo na slici 21. Svaka od njih je zadužena za prikaz vlastitih podataka.

```
<mat-card >
  <mat-card-title>Parametri</mat-card-title>
  <mat-card-content>
    <div class="scroll">
      <app-parameters-per-day *ngFor="let param of parameters" [parameters]="param"></app-parameters-per-day>
    </div>
  </mat-card-content>
</mat-card>
<mat-card>
  <mat-card-title>Terapije</mat-card-title>
  <mat-card-content>
      <div class="scroll">
        <app-therapies *ngFor="let therap of therapies" [therapy]="therap"></app-therapies>
      </div>
  </mat-card-content>
</mat-card>
<mat-card>
  <mat-card-title>Pretrage</mat-card-title>
  <mat-card-content>
    <div class="scroll">
      <app-examinations *ngFor="let examin of examinations" [examination]="examin"></app-examinations>
    </div>
  </mat-card-content>
</mat-card>
/mat-card>
```

Slika 21 TemperatureListComponent.html

4.3. Implementacija mobilne aplikacije

Programski kod mobilne aplikacije organiziran je u nekoliko mapa. Tako smo u mapu ***model*** smjestili razrede koji modeliraju informacije o pacijentu (`Patient`, `Examination`, `Therapy` itd.). U mapi ***screens*** nalaze se razredi `AdminHomeScreen`, `DoctorHomeScreen` i `TemperatureListScreen`, oni nasljeđuju razred `StatefulWidget`, u sebi pohranjuju podatke koje dobivaju od poslužitelja i predstavljaju glavne cjeline mobile aplikacije koje u sebi enkapsuliraju određene razrede koji sadrže svoju grafičku reprezentaciju vidljivu unutar korisničkog sučelja. Uz razrede koje nam pruža Flutter radni okvir, unutar mape `StatelessWidgets` definirali smo i vlastite razrede koji imaju grafičku reprezentaciju (razredi *widget*) a takvi razredi nasljeđuju razred `StatelessWidget` koji je dostupan unutar Flutter radnog okvira.

Na slici 22 prikazana jeimplementacija razreda `DoctorHomeScreen` koji nasljeđuje razred `StatefulWidget`, što znači da unutarnja promjena stanja takvog razreda može rezultirati ponovnim kreiranjem grafičkog prikaza. Radni okvir Flutter omogućuje korištenje razreda i metoda koji upravljaju modulima mobilnih uređaja. U našem slučaju, korištenje NFC-modula mobilnog uređaja omogućeno je razredom `NfcManager`. Pozivom njegove metode `startSession` omogućujemo detekciju NFC-oznake, a kao parametar predajemo anonimnu metodu koja će se izvršiti nakon što se NFC-oznaka uspješno očita. Nakon uspješnog očitavanja NFC-oznake naša anonimna metoda ima instancu razreda `NfcTag` koja je programska reprezentacija skenirane oznake. Pomoću metoda razreda `NfcUtility`, čija je implementacija opisana u nastavku, čitamo vrijednost koja je pohranjena na NFC-oznaci i predajemo je razredu `TemperatureListScreen`, što rezultira otvaranjem temperaturne liste pacijenta.

```dart
class DoctorHomeScreen extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return DoctorHomeScrenState();
  }
}

class DoctorHomeScrenState extends State {
  NfcManager nfcManager = NfcManager.instance;
  String tagData = "-";

  @override
  Widget build(BuildContext context) {
    nfcManager.startSession(onDiscovered: (NfcTag tag) async {
      print(NfcUtility.readNdefTagData(tag));
      setState(() {
        tagData = NfcUtility.readNdefTagData(tag);
      });

      Navigator.of(context).push(MaterialPageRoute(
        builder: (_) {
          return TemperatureListScreen(tagData);
        },
      ));  // MaterialPageRoute
    });

    return Scaffold(
      appBar: AppBar(
        title: Text('Doktor home screen'),
      ),  // AppBar
```

Slika 22 Razred DoctorHomeScreen i DoctorHomeScreenState

Razred `TemperatureListScreenState` prikazan na slici 23 predstavlja stanje razreda `TemperatureListScreen`. Konstruktor razreda prima identifikator kreveta koji se koristi za dohvat podataka o temperaturnoj listi pacijenta. Razred na slici enkapsulira razrede `PatientParameters`, `PatientExaminations` i `PatientTherapies` te im šalje odgovarajuće podatke ako je temperaturna lista uspješno dohvaćena s poslužitelja.

```
TemperatureListScreenState(this.bedId);

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Temperaturna lista pacijenta'),
    ),  // AppBar
    body: _isLoading
        ? Center(
            child: CircularProgressIndicator(),
          )  // Center
        : Center(
            child: Column(
              children: [
                PatientBasicInfo(patient, hospitalization, temperatureList),
                Container(
                  height: 405,
                  child: SingleChildScrollView(
                    child: Column(
                      children: [
                        PatientParameters(parametersList, getTemperatureList,temperatureList.id),
                        PatientExaminations(examinationList,getTemperatureList,temperatureList.id),
                        PatientTherapies(therapyList,temperatureList.id,getTemperatureList),
                      ],
                    ),  // Column
                  ),  // SingleChildScrollView
                )  // Container
              ],
            ),  // Column
          ),  // Center
  );  // Scaffold
```

Slika 23 Razred TemperatureListScreenState

Prilikom izrade sustava za upravljanje temperaturnim listama korištene su NFC-oznake koje podržavaju rad s formatom podataka NDEF. Na slici 24 vidimo razred `NfcUtility` s metodama `readNdefTagData` i `createNdefMessage` koje pomažu za čitanje vrijednosti s NFC-oznake i stvaranja zapisa tipa `NdefMessage` koji odgovara formatu podataka koji koriste NFC-oznake.

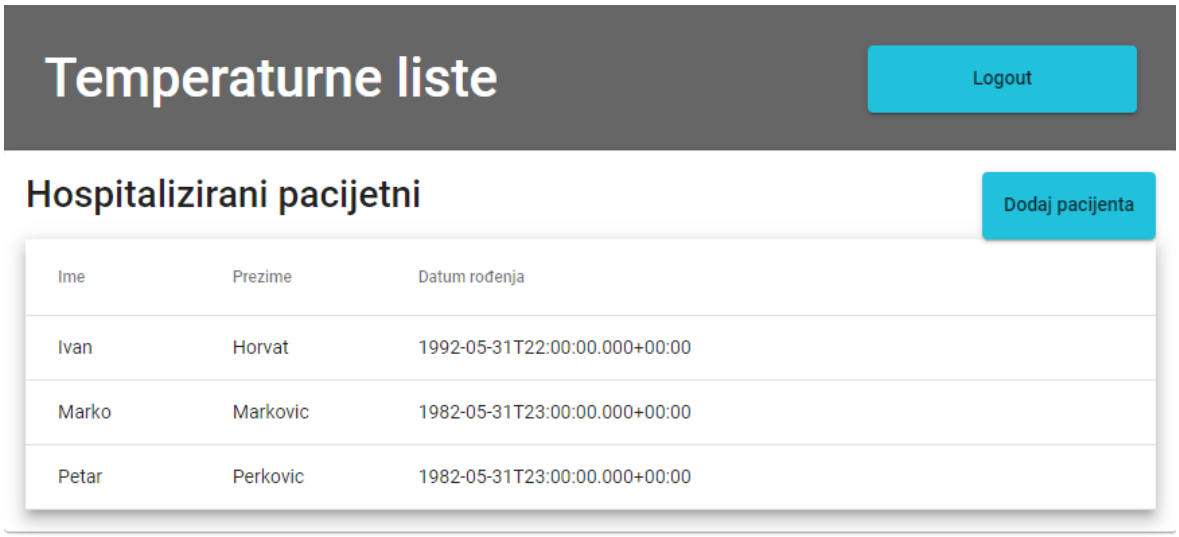
```dart
class NfcUtility {
  static String readNdefTagData(NfcTag tag) {
    Ndef ndef = Ndef.from(tag);
    String result = String.fromCharCodes(ndef.cachedMessage.records[0].payload);

    return result.substring(3, result.length);
  }

  static NdefMessage createNdefMessage(String text) {
    NdefMessage message = NdefMessage([
      NdefRecord.createText(text),
    ]);  // NdefMessage
    return message;
  }
}
```

Slika 24 Razred NfcUtility

4.4. Korisničko sučelje

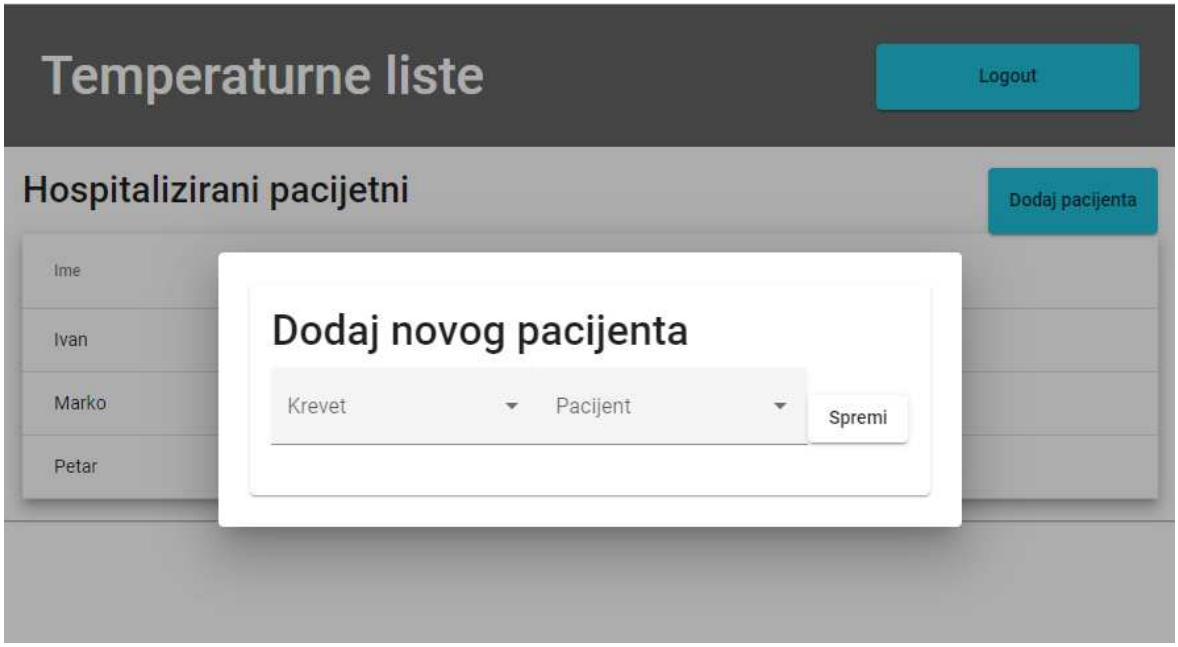
4.4.1. Korisničko sučelje web aplikacije

Na slici 25 vidi se početna stranica vidljiva korisniku s ulogom doktor nakon prijave u sustav, a glavna joj je komponenta tablica u kojoj se nalazi popis svih trenutno hospitaliziranih pacijenata.



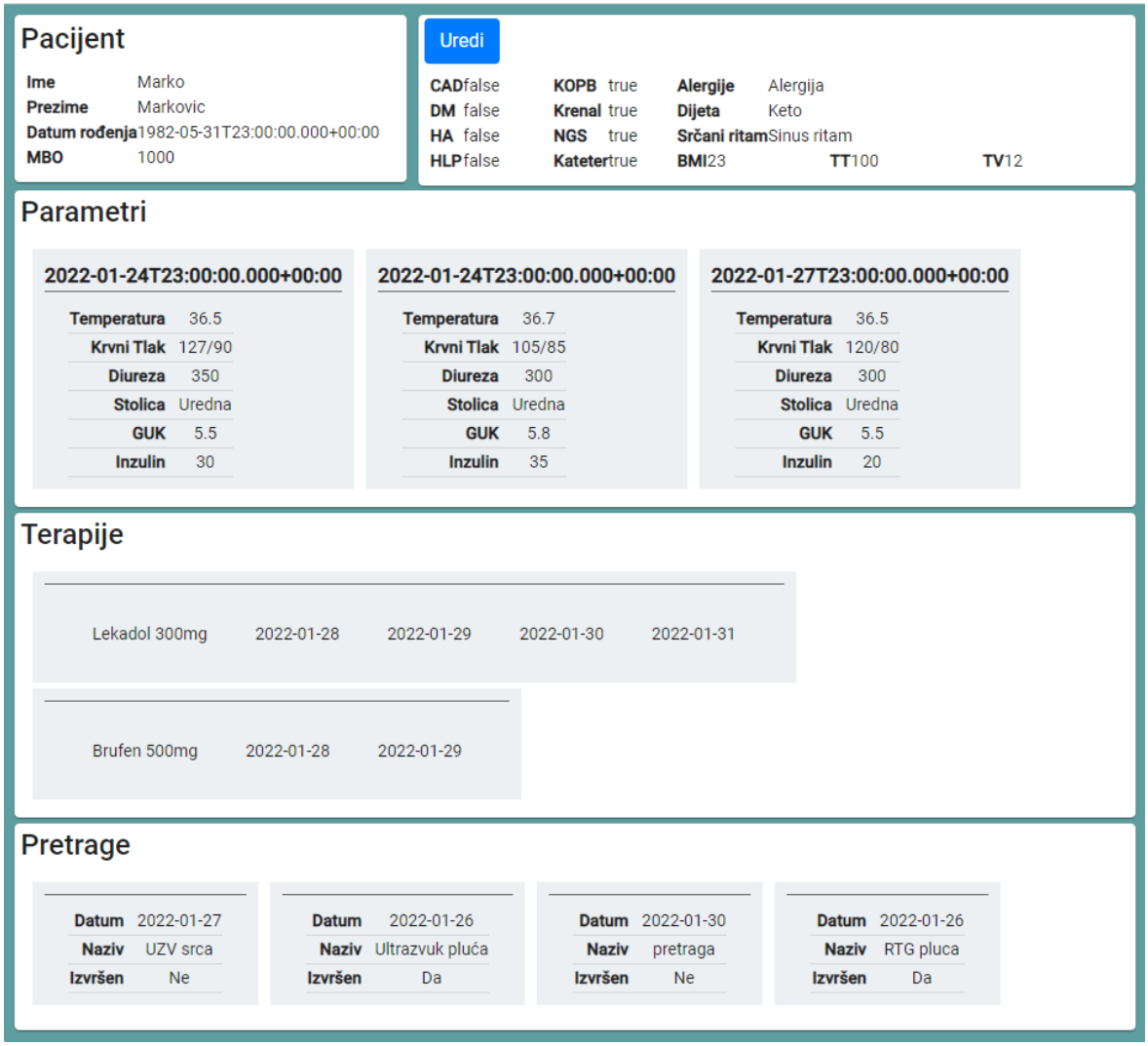
Slika 25 Početna stranica web aplikacije

Korisnik na početnoj stranici ima mogućnost odjave iz sustava kao i dodavanje novog pacijenta. Pritiskom na dugme **dodaj pacijenta** korisniku se otvara komponenta vidljiva na slici 26, unutar koje se nalaze dva padajuća izbornika. Jedan služi za odabir kreveta na koji se želi smjestiti pacijenta a sadrži listu svih postojećih kreveta unutar odjela, dok drugi služi za odabir pacijenta kojeg se hospitalizira. Nakon pritiska na dugme **spremi**, prikazana komponenta se zatvara a novododana hospitalizacija je vidljiva unutar tablice.



Slika 26 Dodavanje novog pacijenta

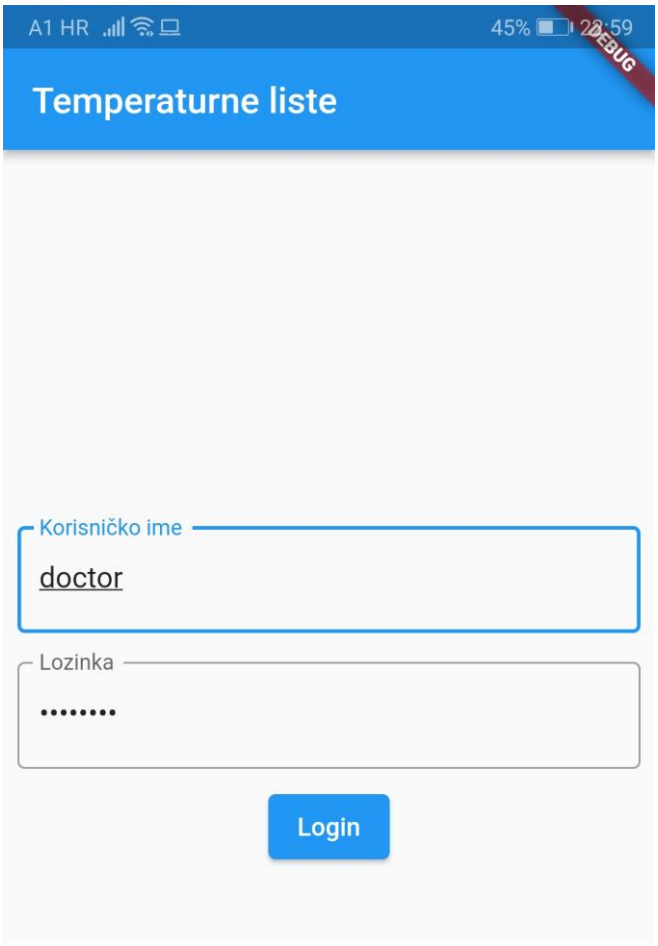
Klikom na pojedini redak u tablici korisniku se otvara prikaz temperaturne liste odabranog pacijenta, vidljivo na slici 27. Prikaz temperaturne liste podijeljen je na nekoliko komponenti. Prikaz osobnih podataka pacijenta, informacije o pacijentovom stanju (alregije, bmi itd.) i podatke koji sadrže vremensku komponentu, parametri, terapije i pretrage.



Slika 27 Prikaz temperaturne liste

4.4.2. Korisničko sučelje mobilne aplikacije

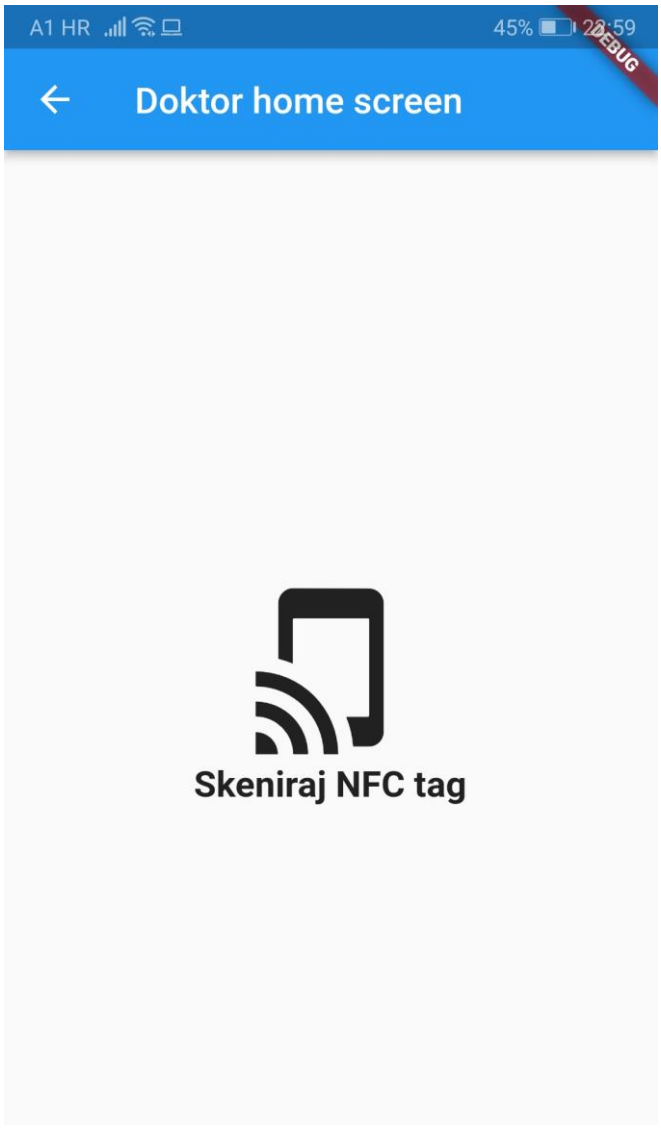
Prilikom pokretanja mobilne aplikacije prikazuje se zaslon vidljiv na slici 28, gdje korisnik unosi podatke potrebne za prijavu u sustav (korisničko ime i lozinka).



Slika 28 Mobilna aplikacija – prijava u sustav

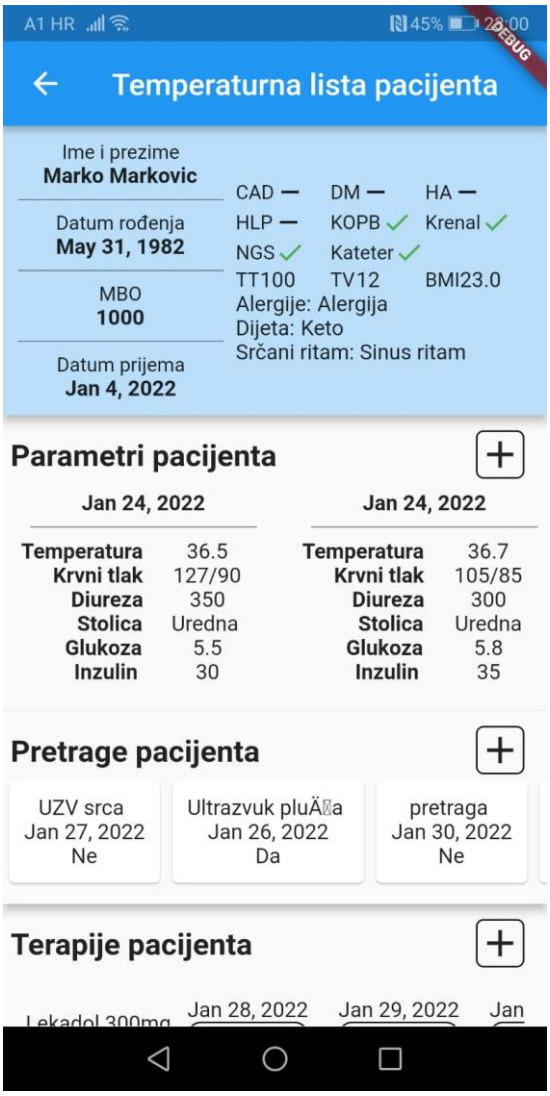
Nakon uspješne prijave, korisniku s ulogom doktor prikazuje se zaslon s porukom o skeniranju NFC-oznake vidljiv na slici 30. Prilikom prikaza navedenog zaslona, aplikaciji je omogućeno skeniranje NFC-oznake, s tim da je prethodno potrebno unutar postavki mobilnog uređaja potrebno omogućiti korištenje NFC-modula, kao što je prikazano na slici 29.

Slika 29 NFC postavke na Android uređaju



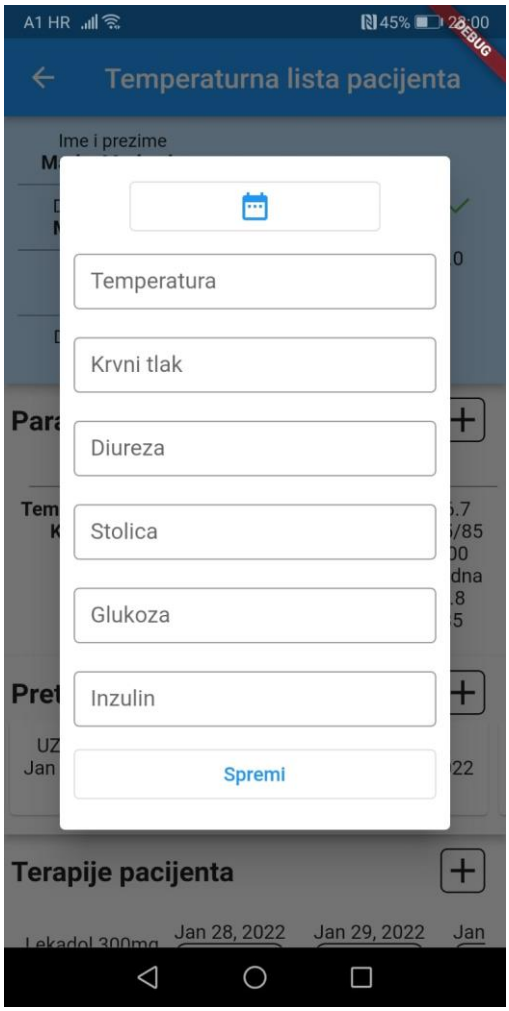
Slika 30 Mobilna aplikacija – početni zaslon

Nakon uspješnog skeniranja NFC-oznake korisniku se otvara zaslon prikazan na slici 31, koji sadrži podatke o temperaturnoj listi pacijenta koji se nalazi na krevetu čiju je oznaku skenirao. Korisnik dalje ima mogućnost dodavanja parametara pacijenta, terapija i pretraga.



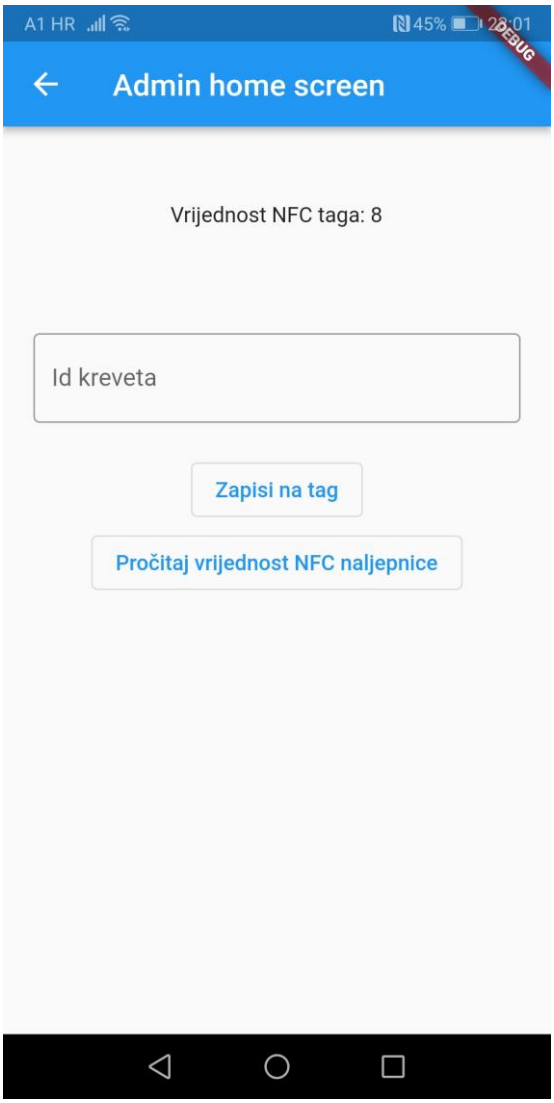
Slika 31 Mobilna aplikacija – prikaz temperaturne liste

Korisniku se pritiskom na dugme u gornjem desnom kutu pojedine komponente otvara prozor s odgovarajućim poljima za unos podataka, ovisno o komponenti koja se dodaje. Tako na slici 32 vidimo prozor koji se prikaže korisniku prilikom dodavanja novih parametara.



Slika 32 Mobilna aplikacija – unos parametara pacijenta

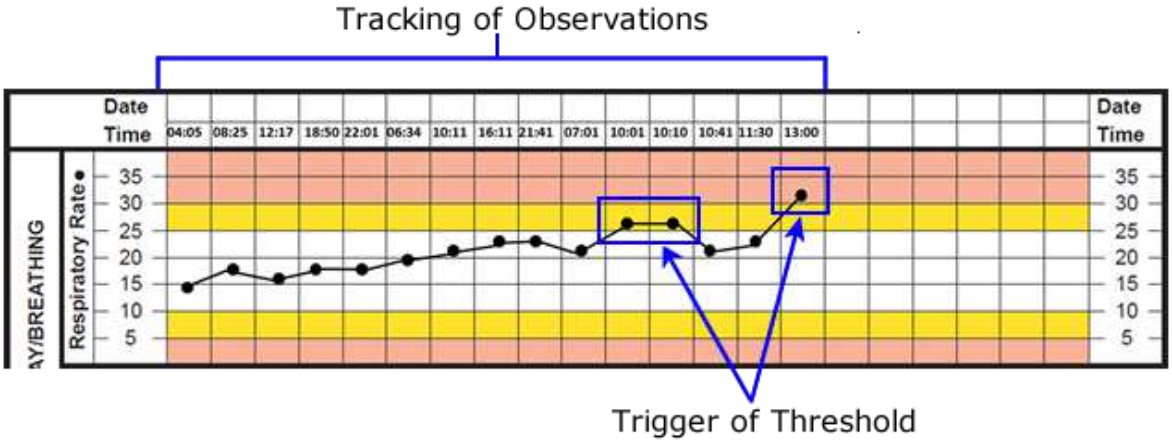
Za razliku od početnog zaslona korisnika s ulogom doktor, korisnik s ulogom administratora ima nešto drugačiji prikaz, vidljivo na slici 33. Administrator na početnom zaslonu ima dvije mogućnosti. Prva je zapisivanje vrijednosti na NFC-oznaku, prije čega je potrebno unijeti identifikator kreveta koji želi pohraniti na NFC-oznaku, stisnuti odgovarajuće dugme na zaslonu te potom prisloniti mobilni uređaj na oznaku. Prilikom skeniranja NFC-oznake mobilni uređaj isporučit će zvučni ili vibrirajući signal, što korisnika informira o uspješnoj komunikaciji s oznakom. Druga je mogućnost čitanja vrijednosti pohranjene na NFC-oznaci, prije koje korisnik također mora stisnuti odgovarajuće dugme, a nakon uspješnog skeniranja, vrijednost pohranjena na oznaci mu se prikazuje na zaslonu.



Slika 33 Mobilna aplikacija – zaslon za rukovanje krevetima

5. Slični postojeći sustavi za evidenciju temperaturnih lista

Novi Južni Wales, savezna država na jugoistoku Australije, u sklopu javnog zdravstvenog sustava razvila je sustav **Between the Flags** s ciljem praćenja i pravovremenog reagiranja prilikom pogoršanja zdravstvenog stanja pacijenta. Unutar temperaturnih lista ugradili su takozvani kriterij pozivanja (engl. *calling critera*) na temelju kojeg se aktivira sustav kliničkog odgovora na hitne slučajeve (engl. *Clinical Emergency Response System* – CERS), koji pak alarmira odgovarajuće medicinsko osoblje da procijeni pacijentovo zdravstveno stanje. Kriterij pozivanja implementirali su kao prikaz pojedinih zona na opservacijskom dijagramu. Svaka zona ima određeni raspon vrijednosti a predstavlja pojedinu razinu ugroze pacijenta. Tako žuta zona zahtijeva da medicinsko osoblje procijeni stanje pacijenta (engl. *clinical review*) dok crvena zona zahtijeva brzi odgovor (engl. *rapid response*) CERS-ovog tima. Grafički prikaz temperaturne liste prikazuje se na monitorima koji se nalaze u bolničkim sobama, a medicinski tehničari unose podatke prilikom mjerenja parametara. Sustav sadrži mehanizam dojave te upozorava medicinskog djelatnika o unesenoj vrijednosti parametra ako se nalazi izvan dozvoljenih granica. Uneseni parametri se zapisuju unutar elektroničkog zdravstvenog zapisa pacijenta (engl. *Electronic Medical Record* – EMR) [8].



Slika 34 Grafički prikaz opservacijskih parametara [8]

Zaključak

U današnje vrijeme jedan od najvrednijih resursa za napredak, kako u znanstvenom vidu tako i u poslovnom aspektu, su podaci. Prvi korak k iskorištavanju podatka je upravo njihovo prikupljanje i adekvatno pohranjivanje. Današnji način spremanja podataka na temperaturnim listama u papirnatom obliku, koji se provodi unutar hrvatskog zdravstvenog sustava, nije pogodan za njihovo efikasno iskorištavanje. Način pohrane opservacijskih podataka koji je implementiran u sustavu opisanom u ovome radu omogućuje njihovo bolje iskorištavanje, kako za samo liječenje pacijenta tako i za daljnja istraživanja koja bi se mogla provoditi na temelju prikupljenih podatka tijekom hospitalizacije pacijenta. Osim spremanja podataka, važna prednost ovakvog sustava je i njihov prikaz u digitalnom obliku. Digitalni prikaz temperaturnih lista uvelike olakšava iskorištavanje prikupljenih podataka, a njihova pohrana daje mogućnost prikaza temperaturne liste u prikladnom grafičkom korisničkom sučelju na webu.

Literatura

[1] M. T. Chatterjee, J. C. Moon, R. Murphy i D. McCrea, »The “OBS” chart: an evidence based approach to re-design of the patient observation chart in a district general hospital setting | Postgraduate Medical Journal,« BMJ Publishing Group Ltd, 20 1 20022. [Mrežno].

[2] »Representational state transfer – Wikipedia,« Wikimedia Foundation, 6 1 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Pokušaj pristupa 30 1 2022].

[3] »The DAO Pattern in Java,« Tarnum Java SRL, 10 11 2021. [Mrežno]. Available: https://www.baeldung.com/java-dao-pattern. [Pokušaj pristupa 30 1 2022].

[4] »JDBC Drivers,« GeeksforGeeks, 9 2 2020. [Mrežno]. Available: https://www.geeksforgeeks.org/jdbc-drivers/. [Pokušaj pristupa 30 1 2022].

[5] »Spring Boot Reference Documentation,« Spring, [Mrežno]. Available: https://docs.spring.io/spring-boot/docs/2.6.3/reference/htmlsingle/. [Pokušaj pristupa 30 1 2022].

[6] »Intro to Inversion of Control and Dependency Injection with Spring,« Tarnum Java SRL, 27 1 2022. [Mrežno]. Available: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring#the-spring-ioc-container. [Pokušaj pristupa 30 1 2022].

[7] »Flutter documentation,« Google, [Mrežno]. Available: https://docs.flutter.dev/. [Pokušaj pristupa 30 1 2022].

[8] »Between the Flags,« NSW Government, [Mrežno]. Available: https://www.cec.health.nsw.gov.au/keep-patients-safe/deteriorating-patient-program/between-the-flags. [Pokušaj pristupa 1 2 2022].

Sažetak

U radu „Web i mobilno rješenje za rukovanje temperaturnim listama zasnovano na tehnologiji NFC“ najprije su prikazani nedostatci trenutno korištenog načina pohrane temperaturnih lista čime je ustanovljena motivacija. Potom je predstavljena ideja njihove pohrane u elektroničkom obliku pri čemu je opisan način unosa i prikaza temperaturnih lista te njihovo dohvaćanje korištenjem tehnologije NFC. Konačno, detaljno je opisana implementacija ovog sustava koristeći suvremene web i mobilne radne okvire Spring, Angular i Flutter.

Ključne riječi: temperaturna lista, podaci pacijenta, opservacijski podaci, web aplikacija, mobilna aplikacija, Flutter, Spring, Spring boot, Java, Angular

Summary

The paper "Web and mobile solution for handling temperature sheets based on NFC technology" first shows the shortcomings of the currently used method of storing temperature sheets, which establishes the motivation. Then, the idea of their storage in electronic form is presented, describing the method of entering and displaying temperature sheets and retrieving them using NFC technology. Finally, the implementation of the system is described in detail using modern web and mobile frameworks Spring, Angular and Flutter.

Keywords: temperature list, patient data, observation charts, web application, mobile application, Flutter, Spring, Spring boot, Java, Angular